01TXFSM - Machine Learning and Deep Learning

Final Project
First Person Action Recognition

Eros Fanì - s269781
Politecnico di Torino
eros.fani@studenti.polito.it

Gabriele Trivigno - s276807
Politecnico di Torino
gabriele.trivigno@studenti.polito.it

Cristiano Gerbino - s277058
Politecnico di Torino
s277058@studenti.polito.it

Abstract

1. Introduction

The main scope of this project was the field of First Person Action Recognition, which is one of the up-and-coming domains in modern Computer Vision, due to the recent spread of wearable devices, often camera-equipped. Even though action recognition and video analysis in general have been topics of research for quite some time now, this particular task introduces some newly faced challenges such as the strong egomotion component, which is inevitable as cameras are placed directly on the actor. Moreover the First Person point of view lacks some important information about the actor and its pose, showing only a part of the scenario, making inference harder. The most frequently adopted techniques to tackle this issues combine spatial information, taken from RGB frames, and motion clues, extracted in various ways, such as temporal stream networks based on optical flow, attention modules or 3D CNNs. We will discuss in depth some of this methodologies, highlighting in our opinion what are the weaknesses of such approaches, and proposing possible ways to circumvent them.

1.1. Goals

The first goal of the project was to replicate the main results of [6]. Afterwards, we integrated the cited architecture with the self-supervised block proposed in [5], implementing it in 2 different ways.

After trying to replicate these experiments with the hyperparameters setting reported in [6], with the experience acquired we chose other sets of hyperparameters that could potentially fit better our dataset of interest, and so performed a grid search upon them.

At last we have tried to improve the performances of the results of [6] and [5] with some innovative ideas of our own.

1.2. Our contribution

2. Related works

Existing literature has underlined how the must crucial information to encode into features in order to obtain good performances on the task of First Person Action Recognition are the hands motion of the actor, and the appearance of objects manipulated, as well as their interaction. Putting together this

2 fundamental pieces of information is perhaps the most challenging task, and more importantly doing so without complicating too much the architecture, which could lead to high computational costs, cutting of potential on-line applications. [6] proposes a spatial attention mechanism to focus on the region consisting of the object, arguing that objects handled are strongly representative of egocentric activities. To keep track of said objects throughout frames, this spatial attention is paired with a with a convLSTM module, whose purpose is to temporal encode the frame-level features formerly extracted. In addition to this, they also use a temporal network fed with optical flow images, following the purpose of better encoding motion changes in the considered videos. The task is then addressed with a late fusion of the 2 outputs with a dense layer to finally perform the classification step. Whilst reasonably successful, this approach consists of a two stream network, which requires several training stages resulting in a massive number of parameters to train. [5] argues that adding a self-supervised block which leverages motion maps at the output of the CNN backbone can provide the convLSTM module with a richer input that already encloses motion information. They show that in this way the network is able to obtain a meaningful enough representation of both appearance and movement to achieve state-of-the-art results, so that the temporal network is no longer needed. Many other works recognize the importance of slimming the architecture to avoid having to train a 2 stream network. [2] introduces a unified model that aims at representing spatio-temporal relationship using only RGB frames, eliminating the heavy computation time required to extract optical flow, which makes the most difference at test time, once the model has been deployed. In their MFNet they insert motion filter blocks in the middle of CNN layers, that act upon feature maps extracted from shared-networks feed-forwarded by two consecutive input frames. The flow is estimated moving each channel of these feature maps in a different spatial direction $\delta := (\Delta x, \Delta y)$. [4] argues that the procedure described, though advantageous parameter-wise and computation-wise, suffer from inferior performance compared with more powerful two-stream networks. They maintain such performance gap is due to the iterative optimization that standard optical flow methods implement and that [2] failes to reproduce. To overcome this issue they propose a new, fully-differentiable, layer, built to extract flow from any CNN feature map. By learning the flow parameters of a smaller resolution CNN tensor, in an end-to-end fashion together with the CNN training, they claim to achieve the same representational power as traditional flow methods, and even better since flows optimized for activity recognition is different from true optical flow. This last two works introduce a great way to encode motion clues in the features extracted, but none of them combine this information with a temporal representation of those features. Whereas [6] fails to exploit RGB frames to get motion informations beyond appearances. Even though [5] tries to overcome this with the motion segmentation task, we found that their approach can sometimes lacks of precision in identifying the region of interest, since it acts on very high-level features. In this project we propose three new different approaches. First of all, we use an improved version of the self-supervised block adopted in [5]to integrate in the work of [6], eliminating the need of a second network and improving the granularity of the motion encoding using higher resolution maps matched with lower level features. Afterwards we adopt the representation-flow layer proposed by [4] together with a temporal encoding module to better exploit temporal relationships in the features extracted. Finally we experimented with a different approach: encoding the concept of dynamics directly into the convLSTM module, adding a discriminator classifier that has been trained to discern whether it has been fed with static frames, or actual video frames.

3. Methodologies overview

Here we describe the models that we have used to perform our experiments.

3.1. Egornn

Egornn is a CNN-RNN joint architecture presented in [6]. The overall architecture of *Egornn* is shown in Figure 1. The CNN backbone is based on *resnet34*[1], which has five main building blocks: with respect to Figure 1 they are: *Conv*, *Layer1*, *Layer2*, *Layer3* and , *Layer4*. From now on we will refer to these blocks respectively *conv1*, *conv2*, *conv3*, *conv4* and *conv5*. It is pree-trained for generic image recognition and complemented with an attention mechanism for spatially selective feature extractin that we are now going to describe.

The exit of *conv5* is used to generate a *Class Activation Map* (CAM). Let $f_l(i)$ be the activation of a unit l in the final convolutional layer at spatial location i and w_c^l be the weight corresponding to class

c for unit l. Then the CAM for class c, $M_c(i)$, can be represented as

$$M_c(i) = \sum_l w_l^c f_l(i)$$

Computing this using only the winning class results in a saliency map of the image highlighting the region in which this object is contained. They argue that this is enough to focus on the object handled by the observer, but we are going to show that this is not always the case. Finally the CAM is converted to a probability spatial map using softmax along the pixel, and multiplied again with the *conv5* output to get the final spatial attention maps.

$$f_{SA}(i) = f(i) \odot \frac{e^{M_c(i)}}{\sum_{i'} e^{M_c(i')}}$$

where $f(i)$ represents the output feature from the final convolutional layer of ResNet-34 at a spatial location i, $M_c(i)$ is the CAM obtained using the winning class c, $f_{SA}(i)$ is the image feature after spatial attention is applied and $\odot$ represents the Hadamard product.

Afterwards the spatial attention is used to feed e convLSTM module that aims at getting a spatial-temporal encoding of the frame-level features extracted by the CNN backbone. It takes into account, for each frame i, both the output of the SAM for the layer i and the output of the ConvLSTM for the layer $i-1$, constituting a recurrent structure This allows the network to learn relationship between spacial and temporal changes, though we argue that what this approach really does is memorize the appearance of an object throughout time, lacking the motion information, and we wll discuss more about this later. ConvLSTM is a RNN, and the difference with the standard LSTM is that it allows to track the spatial features thanks to its memory tensor, instead of the standard vectorial state. So all the multiplication with input, forget and output gates become convolutions. For each cell the equations are the following :

$$\begin{aligned} i_t &= \sigma(w_x^i * f_{SA} + w_h^i * h_{t-1} + b^i) \\ f_t &= \sigma(w_x^f * f_{SA} + w_h^f * h_{t-1} + b^f) \\ \tilde{c_t} &= tanh(w_x^{\tilde{c}} * f_{SA} + w_h^{\tilde{c}} * h_{t-1} + b^{\tilde{c}}) \\ c_t &= \tilde{c_t} \odot f_{SA} + c_{t-1} \odot f_t \\ o_t &= \sigma(w_x^o * f_{SA} + w_h^o * h_{t-1} + b^0) \\ h_t &= o_t \odot tanh(c_t) \end{aligned}$$

where σ is the sigmoid function, i_t, f_t, o_t, c_t and h_t represent the input state, forget state, output state, memory state and hidden state, respectively, of the convLSTM. The trainable weights and biases of the convLSTM are represented using w and b

The last output of the ConvLSTM (the output obtained from the last frame of a particular video) is average pooled and reshaped to obtain a final classification layer with 61 neurons (i.e. the number of classes of our dataset).

3.2. Flow_resnet34

Flow_resnet34 is the second network used by [6]. It is mainly just a *resnet34*, trained separately to work with warp flows. As warp flow are supposed to contain motion clues tied to spatial location, the purpose of this network is to encode the motion changes occurring during an input video. Other than this conceptual difference, it is just and adaption of *resnet34* to accept 10-channel inputs, since it is fed with 5 stacked flow images, each one consisting of a x-flow frame and a y-flow. It can be pointed out that a whole new network is not the most efficient way to extract motion clues, especially since this informations is kept separated from the rest of the network which spatio-temporally encodes features, and only late-fused at the classification step.

3.3. Two stream model

The the proposed approach in [6] is to fuse the *Flow_resnet34*and the *Egornn* resulting in a two stream network. The fusion is proposed in 2 ways: in the naive version, they just average the predictions of the 2 networks; in the second one, a fully connected layer that takes both predictions is added and trained in a fine-tuning stage to get a more proper fusion of the 2 networks contributions. In our experiments we only considered this second approach as the first one does not really exploits having 2 networks. As we already said this late fusion is not very efficient in exploiting the huge representational power that the 2 networks have in terms of number of parameters. Our experiments show that the performance boost obtained joining the two streams is not worth the huge computational weight introduced. We will propose several different approaches to avoid using a second networks.

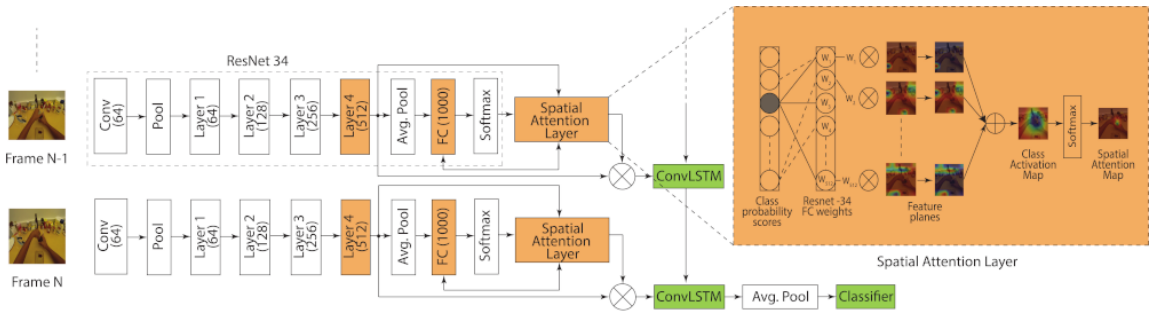


Figure 1: Architecture of *egornn*

3.4. Motion Segmentation branch applied to egornn

This is the architecture proposed in [5], which argues that appearance and motion informations are to be jointly learned, as separating the learning of the 2 is not only heavier but also may lead to lose some of the correlation amongst them without taking into account the spatial-temporal relationships.

The *sparnet* architecture takes on the *egornn* clue of having a CNN backbone feeding a convLSTM module. The objective is achieving the same state-of-the-art result using a single stream network, and moreover leveraging only a limited amount of frames. To achieve this, a secondary task is introduced, consisting in a *Motion Segmentation* (MS) task, formalizable as a binary classification problem on a pixel map, to determine whether each pixel is *moving* or *static*. To implement this task in a self-supervised way, the ground truths for this task are obtained using the *Improved Dense Trajectories* (IDT). This MS block consists in a shallow head attached to the *conv5* output of CNN backbone, so it works with 7x7 feature maps, and after downsampling the channels uses a fully connected layer to produce the predictions for each pixel. In the base version the ground truth is obtained with a simple downsampling of the motion maps from 224x224 to 7x7. The important aspects of this approach are that once the network solved this task at training time, it is supposed to have learned how to extract meaningful features that include motion clues, and so at test time is not needed anymore speeding up inference. Moreover it puts a prior on the feature representation that the network should extract, hence adding a regularization terms which only relies on data. In the end the global training procedure can be formalized as follows: $S_i = H_i, y_{i_{ni=1}}$, where H_i is a set of N times- tamped images $(h_i^k, t_i^k)_{k=1}^N$ uniformly sampled from the video segment. Let also $x = f_M(H|\theta_f, \theta_c)$ be the embed- ding of sample S computed by our model M , where parameters θ_f and θ_c define, respectively, the image embedding and the classification spaces. Finally, let $(p(y|x, \theta_c, \theta_f)$ be the model probability estimator on the embedding x :

$$L_c(x, y) = -\sum_{i=1}^{n} y_i log(p(y|x, \theta_c, \theta_f))$$

$$L_{ms}(x, m) = -\sum_{i=1}^{n}\sum_{k=1}^{N}\sum_{j=1}^{s^2} m_i^k(j) log(l_i^k(j))$$

The pixel losses are summed together (obtaining as result L_{ms}) and then are summed again with the *egornn* loss (L_c). The final loss is used to compute the gradients to update the weights.

$$L(x, y, m|\theta_M) = L(x, y|\theta_f, \theta_c) + L(x, m|\theta_f, \theta_m s)$$

In our implementation, we modify the ego-rnn model to add the MS block to take the outputs of *conv5* and feed them to its shallow head. The last layer has 2 neurons per pixel, each one representing the classes scores (*moving* or *static*). Afterwards the linearized output is used to evaluate the Cross Entropy loss with respect to the ground truth. In PyTorch we sum all the per-pixel losses in a batch and then divide for number of frames and batch size, in order to get the mean loss of the 7x7 maps.

The architecture is shown in Figure 2. We have used this architecture with some granular variations

during our experiments, but the basic blocks are always as shown in Figure 2.

3.4.1 MS task as regression

A possible variation of this task was to implement this self-supervised task as a regressor. In this way, the network output is not anymore a softmax unit that estimates the probability of the possible classes, but is now a single value for each pixel interpreted as the mean of a conditional gaussian

$$\hat{y} = W^T h + b$$
$$p(y|x) = N(y; \hat{y}, I)$$

Where $\hat{y}$ is the linear output units. In this case maximizing the log-likelihood corresponds to minimizing the mean squared error in the training data, so we switched to a Squared Error loss for this task.

$$\hat{y} = W^T h + b$$
$$L(y - \hat{y}) = \sum_{i=1}^{n} (y_i - \hat{y_i})^2$$

After summing the squared errors in our batch, we divide again by batch size and number of frames to get the mean error on a single map.

3.4.2 Our variation on the MS task

Analyzing and comparing the results of this architecture with the one of [6] what we found is that even though it did improve the motion embedding into the features and we show this in the experiments section. Though one thing that this model actually suffered from was the lack of precision in the spatial localisation of motion. For example, looking at misclassified examples, we saw that often an action was mistaken for another if it involved grabbing objects that were close to each other. We argue that a a possible solution to counteract on this, is to move the self-supervised block to the *con3* and *con4* outputs of the CNN backbone. Experimenting with this we found that there actually is a trade of, since the network can definitely benefit from having a finer granularity in the motion maps, but if this map is applied to more low-level features, the representation loses some of its power since there are many more features who are being moved around, and probably a part of that motion is just due to camera shiftings. So trying to get the most efficient trade-of between high-level features, and resolution of the maps, we found that implementing the self-supervised task as a regressor applied at *conv4*, so with 14x14 maps, can be actually quite helpful and achieve a 10 percent performance boost using only 7 frames with respect to the baseline with same number of frames. We chose the regressor over the classifier for the ms block since it gives us the possibility of applying a little trick. Using regression we can obtain the ground truth, instead of downsampling, by taking the mean of the square around the pixel of interest. In this way our ground truths are not anymore binary but actually a grayscale ranging from *static* (0) and *moving* (1).

3.5. Static-dynamic discriminator

Our previous variations of increasing the resolution of the MS task stemmed from the observation that the model suffered from a lack of precision in identifying the motion location. Even with that modifications, the network still had problems in completely discerning between symmetrical actions, like open/close objects. The architecture that we propose to improve upon this situation includes a discriminator at the end of the convLSTM module, trained to discern amongst 2 classes :*static* and *dynamic*. The idea is, instead of struggling to extract feature that encode motion from RGB frames in a CNN which is not built to encode temporal differences, move this task to the convLSTM module, which natively has the capability of representing features that evolve through time. This discriminator is trained jointly with the rest of the network, and is fed with 2 types of inputs: - a first forward pass consisting of the actual output of the CNN, for all frames, labeled as *dynamic* - a second forward pass using n (with n number of frames) times the same feature map. The idea is that in this way the discriminator should penalyze the features that do not encode any motion information, and privilege the others, resulting in a model that is better able to keep track of motions. This task is implemented as a simple binary classification problem with again a Cross Entropy Loss. The final minimization problem, which also includes the MS task from [5], is the following, where t is the label of this new discriminator $t \in$ *static*, *dynamic* , θ_d represents the discriminator classification space, and the rest of the notations is the same as before. The resulting loss to minimize in reported in the last row, and includes weighting parameter α to adjust the impor-

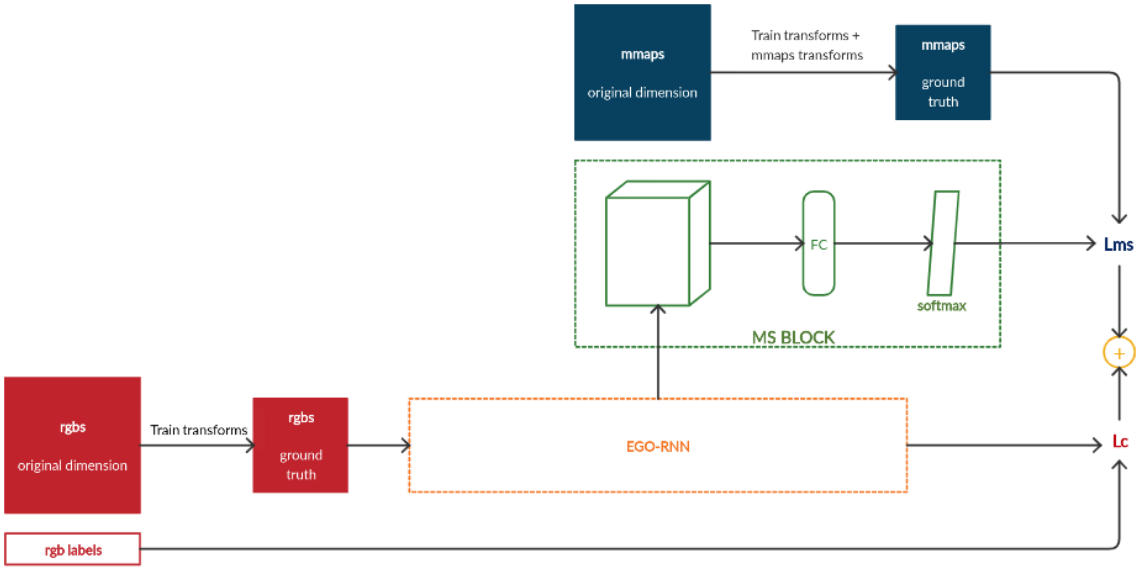


Figure 2: Generic architecture of motion segmentation branch applied to *egornn*

tance to give to the discriminator.

$$L_c(x, y) = -\sum_{i=1}^{n} y_i log(p(y|x, \theta_c, \theta_f))$$

$$L_d(x, t) = -\sum_{i=1}^{n} t_i log(p(t|x, \theta_d, \theta_f))$$

$$L_{ms}(x, m) = -\sum_{i=1}^{n}\sum_{k=1}^{N}\sum_{j=1}^{s^2} m_i^k(j) log(l_i^k(j))$$

$$\begin{aligned} L(x, y, m, t|\theta_M) &= L_c(x, y|\theta_f, \theta_c) + \\ &+ L_{ms}(x, m|\theta_f, \theta_{ms} + \\ &+ \alpha L_d(x, t|\theta_f, \theta_d) \end{aligned}$$

Even though this approach was not able to improve upon our baseline, it did not perform badly ad we report it anyway as it is an elegant and simple solution that may turn out to be useful in cases other than our small GTEA61 dataset.

3.6. Representation Flow layer

Optical flow can be a source of useful features that encode motion and temporal clues. The problem though, it's that it requires an expensive optimization procedure to be obtained, and when used as usual in 2 stream networks renders the inference time quite long, besided making more expensive the training. This idea was based on the works of [2] and [4]. The first one introduces the idea of estimating the flow from directly from RGB frames inside the CNN layers. The second one improves to procedure, making it a bit heavier computationally, but more powerful, embedding it in a fully-differentiable and learnable layer to include in a CNN backbone. The authors of [4] found that inserting this layer at the input performs as poorly as a flow only networks, whereas placing it after the *conv3* block of the backbone resNet achieves the best result. This approach leverages the fact that by estimating flow on a feature map which is low resolution compared to a full images allows to slim the computational cost, and since it is learned in an end-to-end fashion in the training pipeline, is able to extract motion informations that are more related to the features, and so are even better than the standard optical flow. In [4] they apply this to a a CNN backbone, and to a 3D version of it. We argue that including this layer even if it is able to associate the motion information to the feature map that the network extracts, is still lacking a spatio-temporal mapping of the evolution of the features throghout a video. For this reason we included it in our architecture in the following way. We take our frames

through the first 3 residual blocks of our ResNet34 backbone, then we apply the proposed layer to estimate the optical flow from the features map, and then we forward them to the rest of the CNN and afterwards to the convLSTM module. In this scenario we deactivate the spatial attention mechanism, as it was oriented to highlight regions based on the static appearance, and not to work with more motion-oriented features.

3.7. Dataset description

The dataset under analysis is a modified version of GTEA61[1]. The dataset contains the videos in form of frames, and also two kind of preprocessed images: *motion maps* and *optical flows*. The folder schema of the dataset is shown in Figure 5. Videos represent 61 class actions performed by 4 different users (*S1*, *S2*, *S3*, *S4*). Sometimes for some actions more than one video is available. The total number of videos in the dataset is, however, 457, which actually means that it is a quite small dataset.

The optical flow methods try to calculate the motion between two image frames which are taken at times t and $t + \Delta t$ at every voxel position. The warp flow methods try also to remove the motion of the wearable camera. We have two kind of these last representations in our dataset: one computed in the horizontal axis (folder *flow_x_processed*) and one other computed in the vertical axis (folder *flow_y_processed*).

The motion maps are special black-and-white images which represent the spatial location in which the Motion Segmentation task of [5] focuses its attention per each frame. The mmaps present large similarities with the warp flows.

The differences between the kind of available images in our dataset are shown in Figure 6.

3.8. Data cleaning

The dataset was almost clean already from the beginning, but we encountered two problems within it:

- there were hidden useless folders *.DSstore* inside each one of the user folders. These have been removed
- some of the first mmaps of some videos were missing. In these cases we have simply duplicated the second mmap

[1]Georgia Tech Egocentric Activity Datasets: `http://cbs.ic.gatech.edu/fpv/`

4. Experiments

Our nets are always trained on a predefined train set, which includes all and only the videos of the users *S1*, *S3* and *S4*, while validation and test sets coincide and is constituted by all and only the videos of a single user, *S2*. In addition, the weights of the *resnet34* are pretrained on ImageNet. Each model is always validated while it is trained, so for each training phase we selected the weights with the highest accuracy at a particular epoch as the best ones.

Due to Colab limitations of GPU memory, we have only been able to perform experiments on a limited amount of frames (7 or, in less cases, 16). Due to this problem our results should be interpreted not as absolute value of the accuracy, but as a sort of relative value with respect to the number of frames for each video in our batches.

The size of our batches has always been left to 32, as well as the number of hidden units of the convLSTM module, fixed at 512. Our optimization algorithm is always Adaptive moment estimation (ADAM) with the only exception of *flow_resnet34*, for which it is Stochastic Gradient Descent (SGD). When using this last optimizer, the momentum has always been left to 0.9. The scheduler is a MultiStepLR scheduler, which decreases the original learing rate LR by a factor GAMMA at each value of STEP_SIZE.

4.1. Egornn

We have replied some of the same experiments of [6] on the original egornn. We have run each of these experiments three times and then we have averaged the results.

First, we have performed the classification by using the *egornn* without and with the CAM. The training phase has been divided in two parts, as in the original paper:

1. train of ConvLSTM and Classifier (green blocks in Figure 1)
2. train of conv5 (layer4 of *resnet34*), FC(1000),

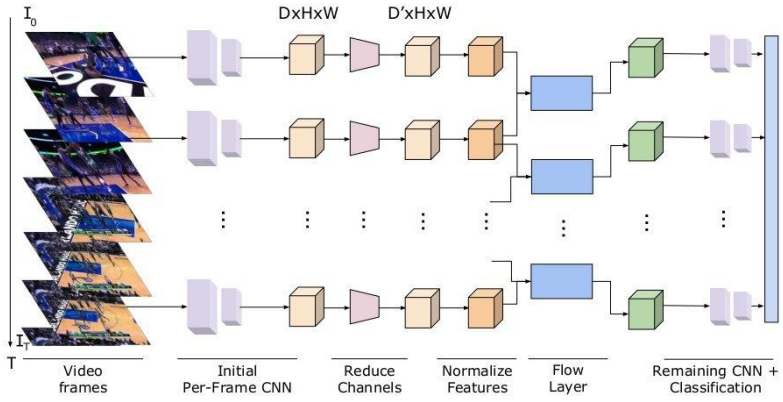


Figure 3: Illustration of a video-CNN with the representation flow layer. The CNN computes intermediate feature maps, that are used as input to the flow layer. The outputs of are used for prediction.

Algorithm 1 Method for the representation flow layer

function REPRESENTATIONFLOW(F_1, F_2)
  $\boldsymbol{u} = 0, \boldsymbol{p} = 0$
  Compute image/feature map gradients
  $\rho_c = F_2 - F_1$
  for n iterations **do**
    $\rho = \rho_c + \nabla_x F_2 \cdot \boldsymbol{u}_x + \nabla_y F_2 \cdot \boldsymbol{u}_y$
    $$\boldsymbol{v} = \begin{cases} \boldsymbol{u} + \lambda\theta\nabla F_2 & \rho < -\lambda\theta|\nabla F_2|^2 \\ \boldsymbol{u} - \lambda\theta\nabla F_2 & \rho > \lambda\theta|\nabla F_2|^2 \\ \boldsymbol{u} - \rho\frac{\nabla F_2}{|F_2|^2} & \text{otherwise} \end{cases}$$
    $\boldsymbol{u} = \boldsymbol{v} + \theta \cdot \text{divergence}(\boldsymbol{p})$
    $\boldsymbol{p} = \frac{\boldsymbol{p} + \frac{\tau}{\theta}\nabla\boldsymbol{u}}{1 + \frac{\tau}{\theta}|\nabla\boldsymbol{u}|}$
  end for
  return $\boldsymbol{u}$
end function

Figure 4: Algorithm to extract optical flow from feature maps of the CNN

Spatial Attention Layer (orange blocks in Figure 1) in addition to the previously listed blocks

The values of the hyperparameters for the first stage are:

LR	1e-3
WEIGHT_DECAY	4e-5
NUM_EPOCHS	200
STEP_SIZE	[25, 75, 150]
GAMMA	0.1

While, for the second stage, they are:

LR	1e-4
WEIGHT_DECAY	4e-5
NUM_EPOCHS	150
STEP_SIZE	[25, 75]
GAMMA	0.1

Then, we have also trained *flow_resnet34* alone. In this case we used only 5 frames per each flow (x and y) due to the fact that for some videos no more than 5 frames were provided.

The values of the hyperparameters in this case are:

LR	1e-2
WEIGHT_DECAY	5e-4
NUM_EPOCHS	750
STEP_SIZE	[150, 300, 500]
GAMMA	0.5

At last we performed the two stream training with the following values for the hyperparameters:

LR	1e-2
LR_FLOW	1e-4
WEIGHT_DECAY	5e-4
NUM_EPOCHS	250
STEP_SIZE	[1]
GAMMA	0.99

Where LR is the learning rate of *egornn* and LR_FLOW is the learning rate of *flow_resnet34*.

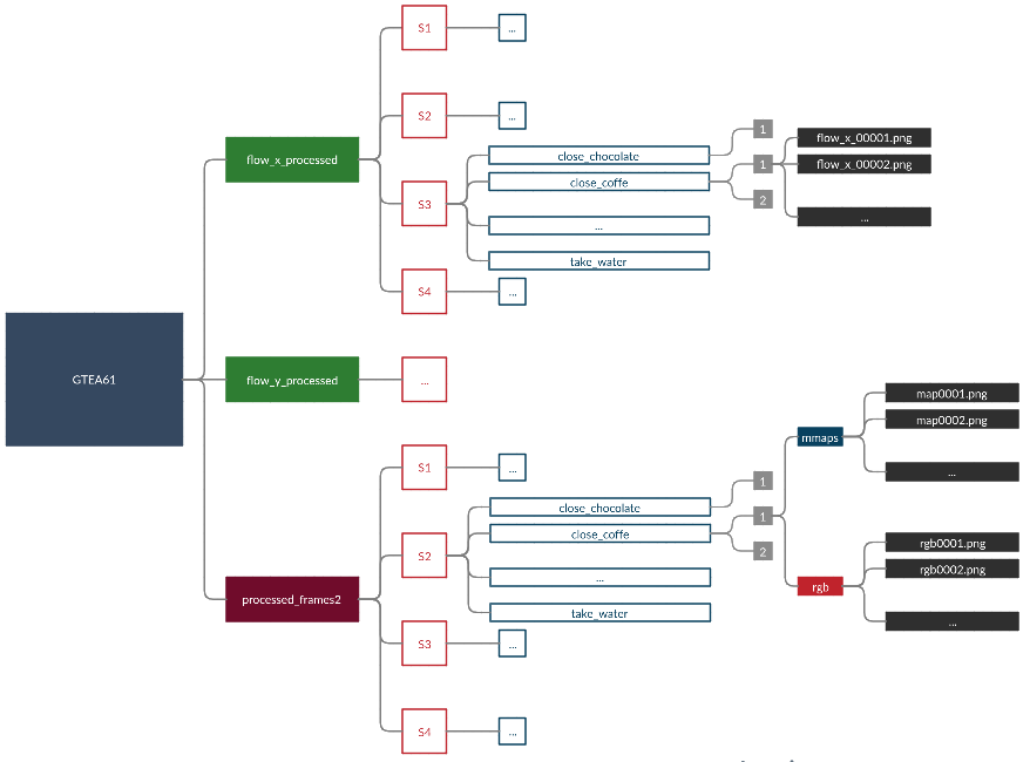


Figure 5: Folder schema of our GTEA61

The summary of our results is shown in Figure 7. From here it raises that the best model is the two-stream (joint train) with 16 frames, followed by EGO-RNN - stage 2 with 16 frames, which is behind the two-stream model by less than 1 point of mean accuracy. Due to the averaging between three identical runs we can rely on this result and assert that the contribution of *flow_resnet34* slightly increases the performances, but also that the most of the contribution is given by *egornn*.

In Figure 8 and Figure 9 are shown respectively the validation accuracy and the validation loss by epoch of one random extracted run per each one of the attempts with 16 frames (5 in case of *flow_resnet34*) and only for the stage 2 when a two stage training is requiered.

From Figure 8 and Figure 9 is even more evident that *flow_resnet34* is highly inefficient alone, and that the results with the CAM are heavily better than the results without the CAM (higher accuracy and lower loss at every epoch). The two-stream model requieres more time to get high accuracies, and overall it seems to have the same behaviour of *egornn* when at full capacity, but it is noisier and so it is easier that for some epoch the accuracy is higher.

In Figure 10 and Figure 11 are shown the CAMs from a couple of video frames exctracted from *egornn*. As we can observe, the precision is not so high: the CAM often miss to focus on the motion.

4.2. Motion Segmentation branch applied to egornn

The wrong focus of the CAMs highly justifies the implementation of the self-supervised task which exploits the motion maps. First of all we have replied the same experiment of [5] (the one in which *egornn* is the Action Recognition Block). The hyperparameters used are:

Figure 6: Types of images in our dataset. In this example is shown a sample of images from the *close_chocolate* action. From the left column to the right column: rgbs, warp flows x, warp flows y, motion maps

Configurations	Frames	Mean accuracy
EGO-RNN without CAM - stage 1	7	29.89
EGO-RNN without CAM - stage 1	16	27.87
EGO-RNN without CAM - stage 2	7	50.00
EGO-RNN without CAM - stage 2	16	50.57
EGO-RNN - stage 1	7	41.38
EGO-RNN - stage 1	16	46.84
EGO-RNN - stage 2	7	58.91
EGO-RNN - stage 2	16	65.52
flow_resnet34	5	46.26
two-stream (joint train)	7*	57.76
two-stream (joint train)	16*	66.38

Figure 7: Summary of the results over different configurations. Each value of the mean accuracy is the mean of the accuracies over three identical experiments. *the number of frames refers to the *egornn* branch (for the flow_resnet34 branch the number of frames is always 5)

LR	1e-3
WEIGHT_DECAY	4e-5
NUM_EPOCHS	150
STEP_SIZE	[50, 100]
GAMMA	0.1

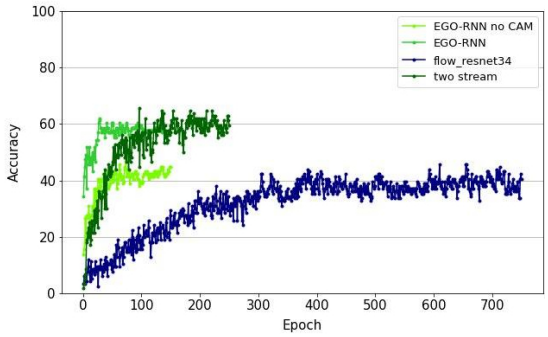


Figure 8: Validation accuracy by epoch of one random extracted run for the four most interesting training configurations

for the stage1, while they are the following:

LR	1e-4
WEIGHT_DECAY	4e-5
NUM_EPOCHS	150
STEP_SIZE	[25, 75]
GAMMA	0.1

for the stage2. We have decreased the number of epochs for the stage1 because we have observed that going too far with the epochs, with the loss

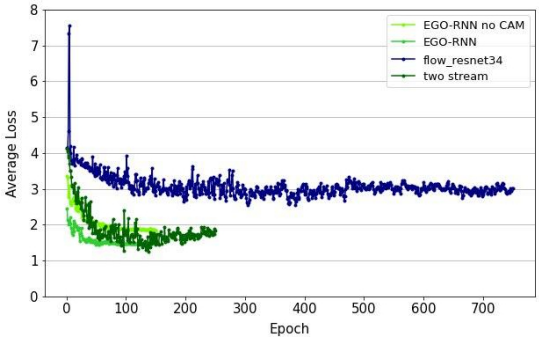


Figure 9: Validation loss by epoch of one random extracted run for the four most interesting training configurations. The losses are the average of each batch loss within a single epoch

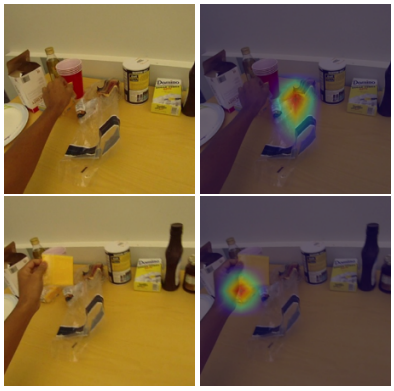


Figure 10: Example of CAMs from take_cheese action

decreased after the steps of the optimizer, would be meaningless and does not give any significant result.

The results with the same values for the hyperparameters are shown in Figure 12.

As expected, the best performances are achieved after the second stage of training and with 16 frames.

We have also replied this experiment as a regression problem. [REGRESSION IMPLEMENTATION]

In Figure 13 are shown the results.

Also in this case the highest value of the accuracy

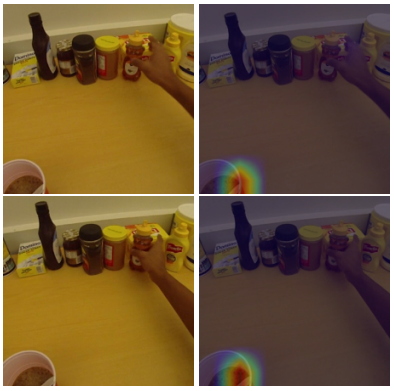


Figure 11: Example of CAMs from take_honey action

Stage	Frames	Mean accuracy
1	7	44.83
1	16	50.86
2	7	60.63
2	16	62.07

Figure 12: Mean accuracies over three identical experiments with the same fixed set of values for the hyperparameters, by varying only the stage of training and the number of frames - classification experiment

Stage	Frames	Mean accuracy
1	7	44.83
	16	50.86
2	7	59.77
	16	66.38

Figure 13: Mean accuracies over three identical experiments with the same fixed set of values for the hyperparameters, by varying only the stage of training and the number of frames - regression experiment

is obtained for 16 frames. The first stage is exactly the same so we have not replied it three times more.

		GAMMA	0.1	0.2	0.5
LR	WEIGHT_DECAY	STEP_SIZE			
1e-04	4e-03	[30, 70]	73.28	64.66	65.52
1e-04	4e-03	[40, 80]	67.24	75.00	67.24
1e-04	4e-03	[50, 100]	67.24	67.24	68.10
1e-04	4e-05	[30, 70]	62.07	68.97	65.52
1e-04	4e-05	[40, 80]	67.24	68.97	66.38
1e-04	4e-05	[50, 100]	65.52	67.24	67.24
5e-04	4e-03	[30, 70]	68.10	64.66	70.69
5e-04	4e-03	[40, 80]	65.52	62.93	63.79
5e-04	4e-03	[50, 100]	62.45	65.52	66.38
5e-04	4e-05	[30, 70]	64.66	65.52	61.23
5e-04	4e-05	[40, 80]	65.52	66.38	62.07
5e-04	4e-05	[50, 100]	68.97	62.45	67.24

Figure 14: Accuracies at various combinations of hyperparameters for the classification strategy

For the second stage we observe that with this combination of values for the hyperparameters the regression performs more than 4 points better than classification with 16 frames, while the accuracies are roughly the same when the number of frames is 7.

We have performed also a complete grid search to improve the performances of the net.

Before starting, it has been executed a brief tuning step for the stage 1 of training, which is the same for both the classification and the regression methodologies. The values of interest have been STEP_SIZE and GAMMA only. From this we obtained that the optimal values for the stage1 are exaclty the same used in the previous experiment.

After this preliminary step, the complete grid search has been executed only on the stage 2, which is the most important one. The number of selected frames is 16, for which we have seen that the performances are better than for 7. We have reduced NUM_EPOCHS to 100 to save computational time (in addition the highest accuracy is rarely obtained after epoch 100). For the classification strategy the results are shown in Figure 14, while for the regression they are shown in Figure 15.

As we can observe the best performances are achieved with:

		GAMMA	0.1	0.2	0.5
LR	WEIGHT_DECAY	STEP_SIZE			
1e-04	4e-03	[30, 80]	62.07	62.93	62.07
1e-04	4e-03	[40, 90]	66.38	68.97	64.66
1e-04	4e-05	[30, 80]	60.34	62.07	67.24
1e-04	4e-05	[40, 90]	62.93	66.38	64.66
5e-04	4e-03	[30, 80]	64.66	61.23	67.24
5e-04	4e-03	[40, 90]	65.52	66.38	71.55
5e-04	4e-05	[30, 80]	68.10	56.03	72.41
5e-04	4e-05	[40, 90]	66.38	65.52	68.10
5e-05	4e-03	[30, 80]	61.21	63.79	62.07
5e-05	4e-05	[30, 80]	61.21	62.07	66.38

Figure 15: Accuracies at various combinations of hyperparameters for the regression strategy

- $LR = 10^{-4}, WEIGHT\_DECAY = 4 \cdot 10^{-3}, STEP\_SIZE = [40, 80]$ for the classification method, for an accuracy of 75.00
- $LR = 5 \cdot 10^{-4}, WEIGHT\_DECAY = 4 \cdot 10^{-5}, STEP\_SIZE = [30, 80]$ for the regression method, for an accuracy of 72.41

In general we can say that the performances are better with the classification method, and this is a surprisingly result based on the observations without the grid. The reason for the previous result was that, as emerges from Figure 14 and Figure 15, the optimal values for the hyperparameters falls in a total different region with respect to the two different methodologies.

In Figure 16 we can observe the effects of the values of the hyperparameters on the accuracy and on the loss of both train and validation and over three different experiments present in the grid search with the classification methodology. The colors of the graphs underline the different learning rate before ad after each step of the optimizer. The steps are shown on the x axis and marked with a vertical gray line. In (a) is shown the best result obtained, while (b) and (c) are representative examples with different tuned values for the hyperparameters. In (a) the loss already starts from a low value and the accuracy already starts from an high value. LR highly influences the starting point of the accuracy and the loss in epoch 1 (higher LR $\rightarrow$ lower initial accuracy and higher initial loss). To show this behaviour it has been picked the example

in (b), which has the same STEP_SIZE but different values for the other hyperparameters. We can observe that, from Figure 14, there is not a tangible difference between the column with GAMMA = 0.1 and the column with GAMMA = 0.2, while it is more evident that with GAMMA = 0.5 the accuracy worsen a bit. So, it has been selected another example, (c), with GAMMA = 0.1 (which is a different gamma with respect to (a)) and the same values of LR and WEIGHT_DECAY (which differs from (a)), and with another STEP_SIZE. The effect of this slight variation on the STEP_SIZE seems to be not so meaningful, while the actual main characters of the different behaviour of the net are LR and WEIGHT_DECAY, together with GAMMA which is good when it is 0.1 as well as when in is 0.2. As last observations:

- is evident from all the three experiments that the train accuracy continue growing also when the validation accuracy starts flattening, and the behaviour is the relative one for the loss
- both the losses and the accuracies are "noisier" when the learning rate is higher
- the global behaviour is strongly influenced by the random batches which feed the net, and so the results should be intended within an error margin

Finally, we can observe examples of CAMs generated by this kind of experiments in Figure 17 and Figure 18.

From these CAMs we can see that the Motion Segmentation Block helps to focus on the movements, but it is sometimes still inaccurate.

4.3. New experiments

4.3.1 The problem of the Motion Maps

The CAMs suggested to us to go in deep with the analysis of the motion maps, trying to improve the attention of the net.

To assert if there is an actual contribution of the Motion Segmentation branch and how much it is significant for the final accuracy we analyzed the loss on the mmaps with a special attention. An example of this loss is shown in Figure 19, which is highly representative because we have observed from several graphs that the behaviour of this loss is always the same. As we can see, the loss on the train set stops to slightly drop after a bunch of epochs.

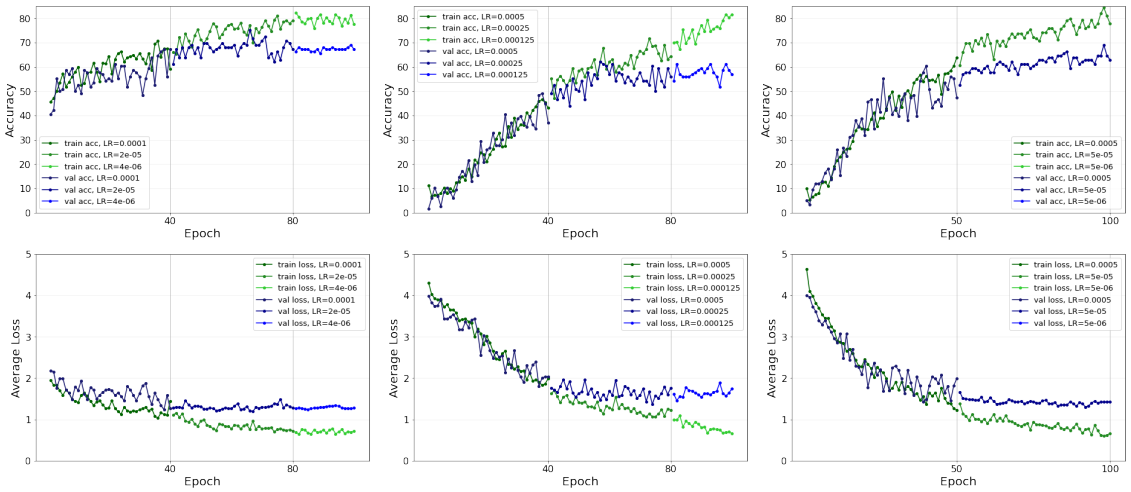
We better studied this behaviour by analyzing the CAMs.

To overcome these problems we have tried to increase the resolution of the downsampled mmaps and to treat the problem as regression problem or binary classification problem, and various combinations of this. When handling with the regression problem, we have also tried to use different kind of downsampled motion maps, with a simple technique developed by us that we called *grid technique* for the sake of comprehensibility. Our base downsampling technique is inspired by [3], but we observed that with a brutal downsampling like this one there is an high risk to loss information. For example, we could take a black pixel contoured by white pixels. This is not good for the net and could bring to misclassification. So, we divided the original 224x224 mmap (obtained after the application of the same kind of trasformations applied to the rgbs without the *normalize* transform) in lxl blocks of $(224/l)$x$(224/l)$ pixels, where l is 7, 14 or 28. The corresponding pixel in the final downsampled mmap is the average of the values of the pixels of its block (so, the image becomes a gray scale image). We applied this last kind of downsampling transformation together with only the regression methodology, due to the particular nature of this image (which in so more easily discretizable).

An example of the described images is shown in Figure 20.

To assert which is the best strategy we have trained the net a first time as in the previouses stages 1, with the last set of values for the hyperparameters already showed for stage1, and we have used these weights as starting point for all the stages 2 of the various implementations. The set of hyperparameters chosen for all the implementations for the stages 2 of this section is the following. The choice is based on the observations that we have already done during the grid search analysis, where we have seen that with these values the net performs good.

LR	1e-4
WEIGHT_DECAY	4e-5
NUM_EPOCHS	150
STEP_SIZE	[30, 80]
GAMMA	0.2



(a)
LR = 1e-4
WEIGHT_DECAY = 4e-3
STEP_SIZE = [40, 80]
GAMMA = 0.2
Highest accuracy = 75.00
at epoch 66

(b)
LR = 5e-4
WEIGHT_DECAY = 4e-5
STEP_SIZE = [40, 80]
GAMMA = 0.5
Highest accuracy = 62.07
at epoch 57

(c)
LR = 5e-4
WEIGHT_DECAY = 4e-5
STEP_SIZE = [50, 100]
GAMMA = 0.1
Highest accuracy = 68.97
at epoch 98

Figure 16: Comparison between three representative experiments in the grid search with the classification method

We have performed three experiments for each one of the selected configurations and then averaged the results, as before. The experiments are summarized in Figure 21.

The first 2 rows of Figure 21 act as baseline for our experiments. They are nothing but the experiments proposed in [5], despite with a lower number of frames due to the already highlighted problems of available memory in Colab. We can observe that:

- the regression over the small 7x7 gray-scale mmaps performed slightly worse than with the 7x7 black-and-white mmaps, but still better than the binary classification baseline version. However, this tell us that the regression technique paired with the gray-scale mmaps is not a good imporvement taken alone
- increasing the resolution from 7x7 to 28x28 is too much. This is not strongly due to the resolution: higher resolutions should give better results. The problem is that to deal with this high resolution downsampled mmaps we have to extract features from preliminary layers, and this bring us to force the training on more low level layers (conv3) to give sense to the Motion Segmentation branch backward phase of gradients updating
- the best trade-off between training the most high level layers possible and to have good high quality representations of the mmaps is for resolution 14x14. Both the techniques performe a lot better then the respective baselines (binary and regression baselines respectively): we get an increase in performances of roughly 7 points of accuracy with respect to the corresponding baseline, which is an awesome result! Considering instead as baseline only the first row, with our best solution we obtained an increase in performances of 13 points of accuracy, which is a substantial improvement!

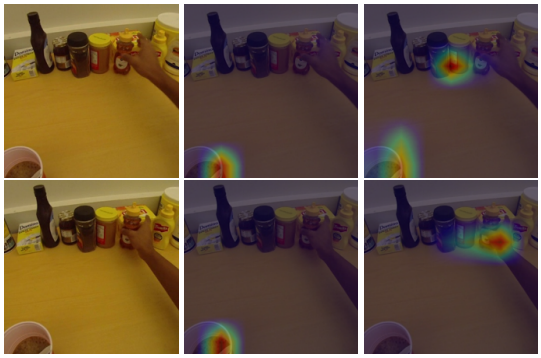


Figure 17: Example of CAMs from take_honey action generated with *egornn* with the Motion Segmentation Branch. The CAMs shown in the center is without MS Block, the CAMs on the right are with the MS Block

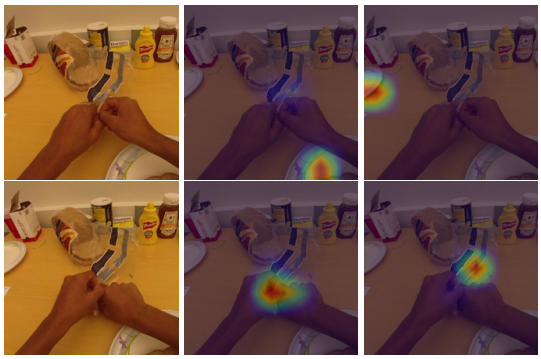


Figure 18: Example of CAMs from take_hotdog action generated with *egornn* with the Motion Segmentation Branch. The CAMs shown in the center is without MS Block, the CAMs on the right are with the MS Block

4.3.2 Static-dynamic discriminator

The reasons for this approach are explained in Section 4.3.2. Also here, due to *Colab out of memory errors*, we had to perform the experiments with 7 frames per video instead than with 16. We have tried four different configurations for the training pipeline, each one tried three times to give consistency to the results:

1. training of *conv5*, *convLSTM* and the final static-dynamic classifier when forwarding the static frames

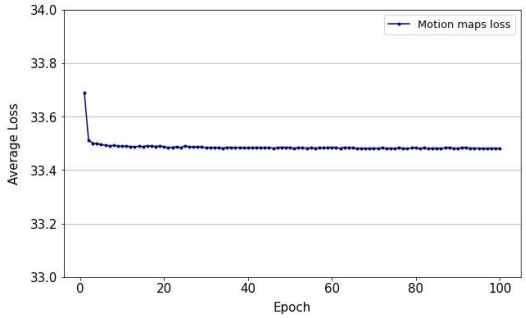


Figure 19: An example of the loss on the classification of the pixels of the downsampled mmaps performed by the Motion Segmentation Branch

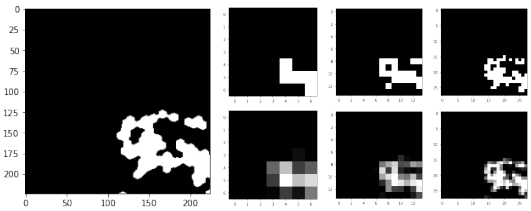


Figure 20: Example of mmap. In the left: original resolution mmap cropped at 224x224. In the right: from left to right mmaps downsampled at 7x7, 14x14 and 28x28 respectively, from top to bottom mmaps classically downsampled and mmaps generated with the "grid technique"

Size	Color	Layer	Classifier	Mean accuracy
7x7	b&w	conv5	binary	56.47
7x7	b&w	conv5	regression	62.07
7x7	gray	conv5	regression	61.21
14x14	b&w	conv4	binary	63.22
14x14	gray	conv4	regression	69.25
28x28	b&w	conv3	binary	54.60
28x28	gray	conv3	regression	52.30

Figure 21: Summary of the mean accuracy values with different combinations of mmaps downsampling techinque and final resolution

conv5 (Y/N)	alpha	Mean accuracy
Y	1.0	55.46
N	1.0	56.03
N	0.5	56.90
N	0.1	64.08

Figure 22: Summary of the mean accuracy values for the Static-Dynamic experiments

2. training of *convLSTM* and the final static-dynamic classifier when forwarding the static frames
3. training *convLSTM* and the final static-dynamic classifier when forwarding the static frames. The static-dynamic loss is multiplied by a constant ALPHA = 0.5
4. training *convLSTM* and the final static-dynamic classifier when forwarding the static frames. The static-dynamic loss is multiplied by a constant ALPHA = 0.1

The reason for the ALPHA constant is to modulate the importance of the Static-dynamic discriminator block.

The Motion Segmentation branch has been used as in [5], and so the downsampled mmaps are black and white 7x7 resolution mmaps and the technique used is the binary classification one.

The results are shown in Figure 23.

The results show that all our attempts have at least a mean accuracy which is comparable to the corresponding baseline (binary classification baseline on the first row of the table in Figure 21). In particular, in the case with ALPHA = 0.1 we get a significant improvement of 8 points of accuracy.

4.3.3 Flow di Gabriele

5. Conclusion

In this paper we presented a further analysis of the methods proposed in [6] and [5]. We integreted these studies with three custom ideas, trying to spot

	Mean accuracy
Ego-rnn two-stream (BASELINE)	57.76
Flow experiment without CAM	51.72
Static-dynamic discriminator	64.08
MMAPS grayscale 14x14 regression	69.25

Figure 23: Summary of the results of our original experiments

the bad points of the proposed methodologies, to improve them with our solutions or joining them with other approaches taken from other researches. The summary of our results is shown in Figure **??**.

Possible lines to follow for further works could try to overcome the still unsolved problems that we have found in our study. These span from the research of a good way to integrate of the warp flows directly in the backbone of the net to a more concrete way to extract motion features. The increases in the performances for the two-stream solution was not so astonishing, and this should bring to try to find innovative solutions in this path. The possibility that we exploited from [4] and [2] was original thought for a ...gabri sistema..., and so...., but ... , so we excluded the possibility to use this in other work based on egocentric action regognition despite of third person action recognition (sistema!!!). Other possibilities for this are... Following the same ideas we also tried to enforce the net to learn the motion features through the Static-Dynamic block with a discretely good result. But the results tell us that our main contribution goes in the direction of the motion maps. We got really high improvements with a simple and elegant idea, based on strong observations on the loss decay of the Motion Segmentation branch. This results give also strength to the study in [5] because the main components of the net that we used are the same of the net in [5]. We hope we could reply (and also improve!) these results in some future work by exploiting a more adequate hardware infrastructure, which would be capable to handle also 16 and 25 frames for each video in our batches. Finally, other attempts that are possible to improve this work is to combine the two-stream methodology with the different downsampling of the mmaps, by treating the motion segmentation branch as regression problem together with the usage of the Static-Dynamic block at the top of *egornn*, and all the other possible combinations of our ideas and of the paper's approaches that have not been explored.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[2] M. Lee, S. Lee, S. Son, G. Park, and N. Kwak. Motion feature network: Fixed motion filter for action recognition, 2018.

[3] D. Pathak, R. Girshick, P. Dollàr, T. Darrel, and B. Hariharan. Learning features by watching objects move, 2017.

[4] A. Piergiovanni and M. S. Ryoo. Representation flow for action recognition, 2019.

[5] M. Planamente, A. Bottino, and B. Caputo. Joint encoding of appearance and motion features with self-supervision for first person action recognition, 2020.

[6] S. Sudhakaran and O. Lanz. Attention is all we need: Nailing down object-centric attention for egocentric activity recognition, 2018.